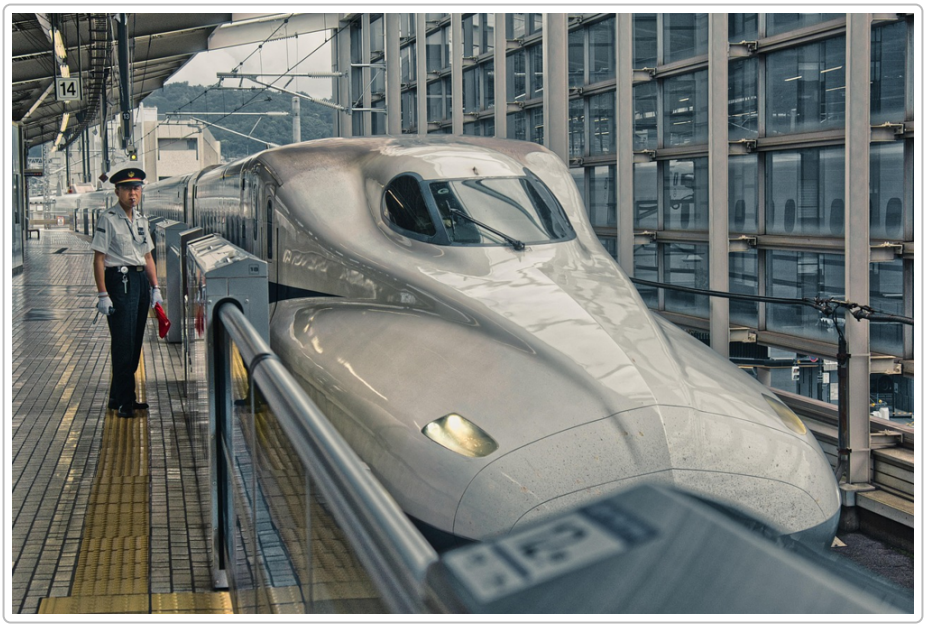
ChatGPT

Proiect BD – Gestionarea biletelor de tren

Descriere model real: Sisteme de gestiune a biletelor de tren pentru operatori publici și privați presupun administrarea trenurilor, stațiilor, traseelor și a tarifelor variabile pentru diferite categorii de pasageri. Modelul cuprinde operatori (companii feroviare), fiecare cu unul sau mai multe trenuri care circulă pe trasee fixate între stații, cu opriri intermediare posibile. Biletele (și rezervările) leagă pasagerii de un traseu/tren specific, într-o anumită dată și categorie tarifară (adult, elev, pensionar etc.), cu prețuri variabile. Sistemul trebuie să stocheze **detalii complete despre trenuri și tarife** (pe traseu, clasă, perioadă) [1], să mențină integritatea datelor (de exemplu, că fiecare bilet are un tren și o categorie validă) și să reflecte regulile de funcționare (un tren poate avea mai multe plecări/rezervări, o stație poate aparține mai multor trasee, etc.). Funcțional, pasagerul alege o stație de plecare, destinație și dată, iar sistemul verifică disponibilitatea trenurilor și afișează tarifele corespunzătoare (pot fi modificate în funcție de clasă, data călătoriei, promoții). Administrativ, se impun reguli precum capacitatea maximă de locuri per tren, validitatea biletelor în funcție de dată, legătura obligatorie între bilete și rezervări, etc. Grafurile de stații și legăturile dintre ele pot fi comparate cu șinele ferate care conectează stațiile într-o rețea coerentă.



Modelul real implică înregistrarea trenurilor care circulă între stații și vânzarea de bilete cu tarife variabile după categorie. Sistemul trebuie să păstreze date precum detaliile trenurilor, orarul și tarifele asociate (pe traseu, clasă și dată) [1]. Toate aceste informații se regăsesc în modelul conceptual: entități pentru `Operator`, `Tren`, `Stație`, `Traseu`, `CategorieTarif`, `Rezervare`, `Bilet` etc., cu reguli de integritate care asigură, de exemplu, că fiecare bilet aparține unui tren și unei rezervări existente, iar prețurile sunt pozitive şi conforme cu categoria tarifară.

2. Constrângeri funcționale și de integritate

- **Chei primare (unicitate):** fiecărei entități i se asociază o cheie primară neambiguă (de exemplu, `OperatorID` pentru `Operator`, `TrenID` pentru `Tren` etc.).
- **Legături (integritate referențială):** de exemplu, `Tren(operator_id)` să fie cheie externă către `Operator(id)`, `Tren(traseu_id)` către `Traseu(id)`, `Traseu(start_statie)` și `Traseu(end_statie)` către `Stație(id)`, `Bilet(rezervare_id)` către `Rezervare(id)` etc., astfel încât să nu existe trenuri sau bilete orfane.
- **Domenii și validări:** constrângeri de tip `NOT NULL` pentru atributele obligatorii, de exemplu numele operatorului, capacitatea trenului; `CHECK` pentru valori permisibile (prețuri > 0, coduri de categorie valide, restricții de vârstă). De asemenea, condiții ca `numar_loc <= Tren.capacitate` sau `Tarif.pret > 0`.
- **Integritate între atribute:** de exemplu, stocarea redundanței minimă – câmpuri precum descrierea unei categorii tarifare nu vor fi repetate în fiecare bilet (evitând anomalii de actualizare).
- **Cardinalități obligatorii:** dacă un tren aparține obligatoriu unui operator, coloana `operator_id` în tabelul `Tren` va fi `NOT NULL`; dacă o stație are întotdeauna un cod unic (`cod_statie` unic), vom impune `UNIQUE (cod_statie)` etc.

3. Definirea entităților și cheilor primare

Entitățile principale și cheile lor primare (**PK**) sunt:
- **Operator**(`OperatorID` PK, Nume, TipOperator (public/privat), ...) – companiile feroviare.
- **Tren**(`TrenID` PK, NumarTren, CapacitateLocuri, TipTren, OperatorID FK, TraseuID FK, ...) – fiecare tren aparține unui operator și unui traseu.
- **Stație**(`StatieID` PK, Nume, Oras, CodStatie, ...) – stațiile la care trenurile opresc.
- **Traseu**(`TraseuID` PK, Denumire, StartStatieFK, EndStatieFK, Distanta, OperatorID FK, ...) – un traseu unică între două stații principale (eventual cu opriri intermediare).
- **Traseu_Statie**(`TraseuID` PK, `StatieID` *PK*, Ordine, TimpSosire, TimpPlecare) – tabel asociativ M:N care listează ordinea și orele de oprire ale fiecărei stații într-un traseu (ordenează stațiile pe traseu). Cheie primară compusă *(TraseuID, StatieID)*.
- **CategorieTarif**(`CategorieID` PK, Descriere, CoeficientReducere, VarstaMin, VarstaMax, ...) – definiții de categorii (adult, copil, student etc.).
- **Tarif**(`TraseuID` PK, `CategorieID` *PK*, Pret) – tabel asociativ M:N ce leagă fiecare traseu cu fiecare categorie tarifară, precizând prețul unitar. Cheie PK compusă *(TraseuID, CategorieID)*.
- **Rezervare**(`RezervareID` PK, DataRezervare, ModalitatePlata, Status, ...) – rezervările realizate (o rezervare poate cuprinde mai multe bilete).
- **Bilet**(`BiletID` PK, RezervareID FK, TrenID FK, CategorieID FK, DataCalatorie, Pret, NrLoc, StatusBilet, ...) – biletele individuale vândute, referință la o rezervare, un tren și o categorie tarifară.

4. Relații între entități (cardinalități)

- **Operator – Tren (1:N):** un operator poate avea multe trenuri, dar fiecare tren are un singur operator.
- **Operator – Traseu (1:N):** un operator deține mai multe trasee; fiecare traseu aparține unui singur operator.
- **Tren – Rezervare (1:N):** un tren poate fi inclus în mai multe rezervări (de obicei la date diferite), o rezervare implicând un singur tren pentru un segment dat.

- **Rezervare – Bilet (1:N):** o rezervare poate avea mai multe bilete (de exemplu pentru grup de pasageri), fiecare bilet aparținând exact unei rezervări.
- **Tren – Bilet (1:N):** un tren poate avea mai multe bilete emise, dar un bilet e pentru un singur tren.
- **Traseu – Stație (N:M):** un traseu cuprinde mai multe stații (opriri); o stație poate face parte din mai multe trasee. Modelat prin tabelul asociativ **Traseu_Statie** (cu attributul „Ordine” care indică ordinea stației în traseu).
- **Traseu – Tarif (1:N):** un traseu poate avea mai multe tarife (pentru categorii diferite); fiecare tarif e asociat unuia dintre trasee. (Din punct de vedere relațional, între **Traseu** și **CategorieTarif** există relația N:M reprezentată de tabelul **Tarif**).
- **CategorieTarif – Tarif (1:N):** o categorie poate apărea în tarife multiple (pe trasee diferite); fiecare tarif corespunde unei singure categorii.
- **Cardinalitate generală:** relațiile se pot 1:1, 1:N sau M:N. De exemplu, una dintre cele mai comune tipuri este 1:N (de ex. „Operator - Tren (1:n)”) sau N:M (cum este „Traseu - Stație (M:N)” prin tabelul asociativ) [2].

În diagrama entitate-relație, entitățile majore sunt reprezentate ca dreptunghiuri (Operator, Tren, Stație, Traseu, etc.), iar relațiile dintre ele (constrângerile de cardinalitate 1:1, 1:N, N:M) apar ca linii/rombi de legătură. De exemplu, relația de tip **1:N** este întotdeauna marcată cu `1` la capătul entității-părinte și `N` la entitatea-copil. Un exemplu conceptual: un operator conduce mai multe trenuri, deci „Operator (1) – Tren (N)”. În diagramă, tabelele asociative precum **Traseu_Statie** sau **Tarif** sunt entități care combină cheile a două entități de bază, indicând relații de tip N:M.

Un diagramă ER generală ar arăta astfel: - **Operator** (1) –– (N) **Tren** –– (N) **Bilet** –– (1) **Rezervare**
- **Tren** –– (N) **Bilet**
- **Traseu** –– (N) **Traseu_Statie** (N) –– (1) **Stație**
- **Traseu** –– (N) **Tarif** (N) –– (1) **CategorieTarif**

Aceste relații asigură integritatea între date și permit extragerea informațiilor complexe (de ex. *“afișează toate biletele vândute pentru trenurile operând pe traseul X”*).

5. Atributele fiecărei entități

• **OPERATOR:**
• `OperatorID` (NUMERIC, PK, nu poate fi NULL, generat automat),
• `Nume` (VARCHAR2(100), NOT NULL),
• `TipOperator` (VARCHAR2(10), de ex. “public”/“privat”, NOT NULL),
• `Adresă` (VARCHAR2(200), poate fi NULL), etc.

• **TREN:**

• `TrenID` (NUMERIC, PK, NOT NULL),
• `NumarTren` (VARCHAR2(10), UNIC, NOT NULL),
• `CapacitateLocuri` (NUMBER, NOT NULL, CONSTRAINT >= 0),
• `TipTren` (VARCHAR2(20), ex. “personale”, “marfă”),
• `OperatorID` (NUMERIC, FK→OPERATOR(OperatorID), NOT NULL),
• `TraseuID` (NUMERIC, FK→TRASEU(TraseuID), NOT NULL).

• **STATIE:**

• `StatieID` (NUMERIC, PK, NOT NULL),
• `Nume` (VARCHAR2(100), NOT NULL),
• `Oras` (VARCHAR2(100), NOT NULL),
• `CodStatie` (VARCHAR2(10), UNIQUE, NOT NULL).

• **TRASEU:**

• `TraseuID` (NUMERIC, PK, NOT NULL),
• `Denumire` (VARCHAR2(100), NOT NULL),
• `Distanta` (NUMBER, distanța totală în km, NOT NULL),
• `StartStatieID` (NUMERIC, FK→STATIE(StatieID), NOT NULL),
• `EndStatieID` (NUMERIC, FK→STATIE(StatieID), NOT NULL),
• `OperatorID` (NUMERIC, FK→OPERATOR(OperatorID), NOT NULL).

• **TRASEU_STATIE (Traseu-Stație):**

• `TraseuID` (NUMERIC, PK*, FK→TRASEU(TraseuID), NOT NULL),
• `StatieID` (NUMERIC, PK*, FK→STATIE(StatieID), NOT NULL),
• `Ordine` (NUMBER, ordinea stației pe traseu, NOT NULL),
• `TimpSosire` (DATE, poate fi NULL dacă nu aplică),
• `TimpPlecare` (DATE).

• *PK compus din (TraseuID, StatieID).*

• **CATEGORIE_TARIF:**

• `CategorieID` (NUMERIC, PK, NOT NULL),
• `Descriere` (VARCHAR2(50), ex. “Adult”, “Student”, NOT NULL),
• `CoefReducere` (NUMBER, ex. 0.0..1.0, NOT NULL),

• `VarstaMin` (NUMBER), `VarstaMax` (NUMBER) – interval de vârstă.

• **TARIF:**

• `TraseuID` (NUMERIC, PK*, FK→TRASEU(TraseuID), NOT NULL),
• `CategorieID` (NUMERIC, PK*, FK→CATEGORIE_TARIF(CategorieID), NOT NULL),
• `Pret` (NUMBER, NOT NULL, PRET > 0).

• *PK compus din (TraseuID, CategorieID).*

• **REZERVARE:**

• `RezervareID` (NUMERIC, PK, NOT NULL),
• `DataRezervare` (DATE, default SYSDATE, NOT NULL),
• `ModalitatePlata` (VARCHAR2(20), ex. “card”, “numerar”),

• `Status` (VARCHAR2(20), ex. “activa”, “anulata”).

• **BILET:**

• `BiletID` (NUMERIC, PK, NOT NULL),
• `RezervareID` (NUMERIC, FK→REZERVARE(RezervareID), NOT NULL),
• `TrenID` (NUMERIC, FK→TREN(TrenID), NOT NULL),
• `CategorieID` (NUMERIC, FK→CATEGORIE_TARIF(CategorieID), NOT NULL),
• `DataCalatorie` (DATE, NOT NULL),
• `Pret` (NUMBER, NOT NULL, PRET > 0),
• `NrLoc` (VARCHAR2(5), ex. “A12”),
• `StatusBilet` (VARCHAR2(10), ex. “valid”, “anulat”).
• Se poate impune `CHECK (Pret > 0)`, `CHECK (DataCalatorie >= SYSDATE)` etc.

6. Diagrama entitate-relație

Diagrama E-R (Entitate-Relatie) pune în evidență vizual entitățile, atributele-cheie (subiniate) și legăturile logice dintre ele (cu cardinalități). De exemplu, în diagrama ER fiecare entitate este reprezentată drept un obiect și fiecare relație are marcate capetele 1 sau N; un traseu este conectat la stații printr-o relație de tip M:N (prin tabelul asociativ *Traseu_Statie*), iar un operator și un tren prin relație 1:N.

Diagrama conceptuală ar arăta, spre exemplu, entitățile menționate, fiecare cu cheia primară subliniată, și liniile ce leagă entitățile cu simbolurile cardinalității. **ER diagram** „arata relatiile între entitatile din baza de date” [3], îndeosebi natura 1:1, 1:N sau N:M. În acest caz, entitățile principale (Operator, Tren, Stație, Traseu, Rezervare, Bilet, CategorieTarif) sunt conectate în diagramă conform descrierii de mai sus. Pentru claritate,

obiectul „Traseu_Statie” apare ca entitate asociativă (legește între Traseu și Stație, având atributul *Ordine*), iar „Tarif” între Traseu și CategorieTarif. O astfel de diagramă ar îndeplini toate constrângerile semantice (de ex. fiecare bilet este legat de un tren și o rezervare valide).

7. Diagrama conceptuală derivată (minim 7 tabele, cu tabel asociativ)

Pornind de la modelul real s-au obținut cel puțin următoarele tabele (scheme conceptuale): - **OPERATOR**(`OperatorID`, Nume, TipOperator, ...)
- **TREN**(`TrenID`, NumarTren, CapacitateLocuri, OperatorID → OPERATOR, TraseuID → TRASEU, ...)
- **STATIE**(`StatieID`, Nume, Oras, CodStatie, ...)
- **TRASEU**(`TraseuID`, Denumire, Distanta, StartStatieID → STATIE, EndStatieID → STATIE, OperatorID → OPERATOR, ...)
- **TRASEU_STATIE**(`TraseuID` → TRASEU, `StatieID` → STATIE, Ordine, TimpSosire, TimpPlecare) – tabel asociativ.
- **CATEGORIE_TARIF**(`CategorieID`, Descriere, CoefReducere, ...)
- **TARIF**(`TraseuID` → TRASEU, `CategorieID` → CATEGORIE_TARIF, Pret) – tabel asociativ pentru tarife.
- **REZERVARE**(`RezervareID`, DataRezervare, ModalitatePlata, Status)
- **BILET**(`BiletID`, RezervareID → REZERVARE, TrenID → TREN, CategorieID → CATEGORIE_TARIF, DataCalatorie, Pret, NrLoc, StatusBilet)

Diagrama conceptuală ar lista aceste entități (nume_substantiv) și atributele lor, cu săgeți/legături care arată asocieri (de regulă scrise cu verbe, ex. „vinde”, „apartine” etc.). Cheile primare apar subliniate, iar asocierile M:N devin tabele asociative (ca „Traseu_Statie” şi „Tarif”).

8. Schemele relaționale corespunzătoare

Trecem tabel cu tabel în relații (normalizate):

- **OPERATOR**(**OperatorID**, Nume, TipOperator, Adresa)
- **TREN**(**TrenID**, NumarTren, CapacitateLocuri, TipTren, OperatorID, TraseuID) – cu chei externe `FK(OperatorID)→OPERATOR`, `FK(TraseuID)→TRASEU`.
- **STATIE**(**StatieID**, Nume, Oras, CodStatie)
- **TRASEU**(**TraseuID**, Denumire, Distanta, StartStatieID, EndStatieID, OperatorID) – `FK(StartStatieID)→STATIE`, `FK(EndStatieID)→STATIE`, `FK(OperatorID)→OPERATOR`.
- **TRASEU_STATIE**(**TraseuID**, **StatieID**, Ordine, TimpSosire, TimpPlecare) – cheie primară compusă (TraseuID,StatieID), `FK(TraseuID)→TRASEU`, `FK(StatieID)→STATIE`.
- **CATEGORIE_TARIF**(**CategorieID**, Descriere, CoefReducere, VarstaMin, VarstaMax)
- **TARIF**(**TraseuID**, **CategorieID**, Pret) – cheie compusă (TraseuID,CategorieID), `FK(TraseuID)→TRASEU`, `FK(CategorieID)→CATEGORIE_TARIF`.
- **REZERVARE**(**RezervareID**, DataRezervare, ModalitatePlata, Status)
- **BILET**(**BiletID**, RezervareID, TrenID, CategorieID, DataCalatorie, Pret, NrLoc, StatusBilet) – cheile externe `FK(RezervareID)→REZERVARE`, `FK(TrenID)→TREN`, `FK(CategorieID)→CATEGORIE_TARIF`.

Aceste scheme asigură *una la una* și *una la multe* exact prin cheile străine, iar asociativele realizează mulți-la-mulți.

9. Normalizarea până la FN3 (1NF, 2NF, 3NF)

Normalizarea urmărește **structurarea relațiilor** pentru a reduce redundanța datelor și a elimina anomalii de inserare, ștergere și actualizare [4]. Practic, se aplică reguli formale pentru a transforma tabelele introduse inițial (uneori ne-normalizate) în forme normalizate: 1NF asigură valori atomice, 2NF elimină dependențele parțiale față de cheia compusă, 3NF elimină dependențele tranzitive.

- **1NF:** fiecare câmp conține o valoare atomică (nu liste sau repetări în aceeași coloană). De exemplu, dacă inițial tabelul *Rezervare* avea câmpuri precum *Pasageri*[listă] sau *StațiiOprite*[listă], le separăm în relații distincte. După 1NF, toate atributele sunt simple (ex. *Traseu_Statie* este creată ca tabel separat în loc să avem o coloană cu șir de stații).

- **2NF:** eliminăm dependențele parțiale. Dacă un tabel are cheie primară compusă și un atribut depinde doar de o parte din cheie, îl extragem într-un alt tabel. *Exemplu:* inițial am fi putut avea un tabel `Tarifuri(TraseuID, CategorieID, Pret, DescriereCategorie)`. Cheia compusă este (TraseuID,CategorieID), însă *DescriereCategorie* depinde doar de `CategorieID`. Pentru 2NF, despărțim: `CategorieTarif(CategorieID,Descriere)` și lăsăm `Tarif(TraseuID, CategorieID, Pret)`.

- **3NF:** eliminăm dependențele tranzitive. Aici fiecare atribut ne-cheie trebuie să depindă direct de cheia primară, nu printr-un alt atribut. În exemplu de mai sus, după despărțire nu rămân dependențe tranzitive, deci am atins 3NF. În general, în 3NF „majoritatea tabelelor sunt libere de anomalii” și de aceea o relație se consideră normalizată când e în 3NF [5].

Prin aceste transformări, fiecare tabel final (cum sunt cele listate la punctul 8) îndeplinește condițiile de 3NF, reducând redundanța. De exemplu, în `REZERVARE` nu există coloane care depind de altceva decât de RezervareID; în `BILET` nu sunt atribute care să depindă de ceva diferit de cheia sa; în `TARIF` doar `Pret` rămâne ca necheie, și depinde pe deplin de cheile (TraseuID, CategorieID).

10. Secvență pentru inserare de date

Pentru a genera automat valori unice la inserarea în tabele, se poate crea o secvență Oracle. De exemplu:

```
CREATE SEQUENCE seq_operator
  START WITH 1
  INCREMENT BY 1
  NOCACHE;
```

Se pot crea secvențe similare pentru fiecare tabel cu cheie numerică (ex. `seq_tren`, `seq_statie` etc.). La inserare, coloana cheii primare primește `seq_operator.NEXTVAL`.

Exemplu de utilizare la inserare:

```sql
INSERT INTO OPERATOR(OperatorID, Nume, TipOperator)
VALUES (seq_operator.NEXTVAL, 'CFR Călători', 'public');
```

11. Crearea tabelelor și inserarea de date

```sql
-- Creare tabele (Oracle SQL)
CREATE TABLE OPERATOR (
  OperatorID    NUMBER PRIMARY KEY,
  Nume          VARCHAR2(100) NOT NULL,
  TipOperator   VARCHAR2(20) NOT NULL
);

CREATE TABLE STATIE (
  StatieID    NUMBER PRIMARY KEY,
  Nume        VARCHAR2(100) NOT NULL,
  Oras        VARCHAR2(100) NOT NULL,
  CodStatie   VARCHAR2(10) UNIQUE NOT NULL
);

CREATE TABLE TRASEU (
  TraseuID      NUMBER PRIMARY KEY,
  Denumire      VARCHAR2(100) NOT NULL,
  Distanta      NUMBER NOT NULL,
  StartStatieID NUMBER NOT NULL,
  EndStatieID   NUMBER NOT NULL,
  OperatorID    NUMBER NOT NULL,
  CONSTRAINT fk_traseu_start FOREIGN KEY (StartStatieID) REFERENCES
STATIE(StatieID),
  CONSTRAINT fk_traseu_end   FOREIGN KEY (EndStatieID)   REFERENCES
STATIE(StatieID),
  CONSTRAINT fk_traseu_op    FOREIGN KEY (OperatorID)    REFERENCES
OPERATOR(OperatorID)
);

CREATE TABLE TREN (
  TrenID          NUMBER PRIMARY KEY,
  NumarTren       VARCHAR2(10) NOT NULL UNIQUE,
  CapacitateLocuri NUMBER NOT NULL CHECK (CapacitateLocuri >= 0),
  TipTren         VARCHAR2(20),
  OperatorID      NUMBER NOT NULL,
  TraseuID        NUMBER NOT NULL,
  CONSTRAINT fk_tren_op   FOREIGN KEY (OperatorID) REFERENCES
OPERATOR(OperatorID),
  CONSTRAINT fk_tren_tr   FOREIGN KEY (TraseuID)   REFERENCES TRASEU(TraseuID)
);
```

```
CREATE TABLE TRASEU_STATIE (
  TraseuID    NUMBER NOT NULL,
  StatieID    NUMBER NOT NULL,
  Ordine      NUMBER NOT NULL,
  TimpSosire  DATE,
  TimpPlecare DATE,
  PRIMARY KEY (TraseuID, StatieID),
  CONSTRAINT fk_ts_traseu FOREIGN KEY (TraseuID) REFERENCES TRASEU(TraseuID),
  CONSTRAINT fk_ts_statie FOREIGN KEY (StatieID) REFERENCES STATIE(StatieID)
);

CREATE TABLE CATEGORIE_TARIF (
  CategorieID    NUMBER PRIMARY KEY,
  Descriere      VARCHAR2(50) NOT NULL,
  CoefReducere   NUMBER NOT NULL,
  VarstaMin      NUMBER,
  VarstaMax      NUMBER
);

CREATE TABLE TARIF (
  TraseuID      NUMBER NOT NULL,
  CategorieID   NUMBER NOT NULL,
  Pret          NUMBER NOT NULL CHECK (Pret > 0),
  PRIMARY KEY (TraseuID, CategorieID),
  CONSTRAINT fk_tarif_tr FOREIGN KEY (TraseuID)    REFERENCES TRASEU(TraseuID),
  CONSTRAINT fk_tarif_ca FOREIGN KEY (CategorieID) REFERENCES
CATEGORIE_TARIF(CategorieID)
);

CREATE TABLE REZERVARE (
  RezervareID    NUMBER PRIMARY KEY,
  DataRezervare  DATE DEFAULT SYSDATE NOT NULL,
  ModalitatePlata VARCHAR2(20),
  Status         VARCHAR2(20)
);

CREATE TABLE BILET (
  BiletID       NUMBER PRIMARY KEY,
  RezervareID   NUMBER NOT NULL,
  TrenID        NUMBER NOT NULL,
  CategorieID   NUMBER NOT NULL,
  DataCalatorie DATE NOT NULL,
  Pret          NUMBER NOT NULL CHECK (Pret > 0),
  NrLoc         VARCHAR2(5),
  StatusBilet   VARCHAR2(20),
  CONSTRAINT fk_bilet_rez FOREIGN KEY (RezervareID) REFERENCES
REZERVARE(RezervareID),
```

```
  CONSTRAINT fk_bilet_tren FOREIGN KEY (TrenID)      REFERENCES TREN(TrenID),
  CONSTRAINT fk_bilet_cat  FOREIGN KEY (CategorieID) REFERENCES
CATEGORIE_TARIF(CategorieID)
);
```

Inserare date (exemple):

```
-- Operatori
INSERT INTO OPERATOR VALUES (1, 'CFR Călători', 'public');
INSERT INTO OPERATOR VALUES (2, 'SNCF', 'privat');
INSERT INTO OPERATOR VALUES (3, 'Metrorail', 'privat');
-- Stații
INSERT INTO STATIE VALUES (10, 'București Nord', 'București', 'BNR');
INSERT INTO STATIE VALUES (11, 'Cluj Napoca', 'Cluj', 'CLJ');
INSERT INTO STATIE VALUES (12, 'Timișoara Est', 'Timișoara', 'TME');
INSERT INTO STATIE VALUES (13, 'Iași', 'Iași', 'IAS');
INSERT INTO STATIE VALUES (14, 'Constanța', 'Constanța', 'CON');
-- Trasee
INSERT INTO TRASEU VALUES (100, 'Nord-Sud', 100, 10, 12, 1);
INSERT INTO TRASEU VALUES (101, 'Cluj-Constanța', 800, 11, 14, 2);
-- Trenuri
INSERT INTO TREN VALUES (1000, 'IR1732', 200, 'Personal', 1, 100);
INSERT INTO TREN VALUES (1001, 'TGV777', 300, 'Rapid', 2, 101);
-- Traseu_Statie (exemplu opriri)
INSERT INTO TRASEU_STATIE VALUES (100, 10, 1, TO_DATE('08:00','HH24:MI'),
TO_DATE('08:05','HH24:MI'));
INSERT INTO TRASEU_STATIE VALUES (100, 12, 2, TO_DATE('10:30','HH24:MI'),
TO_DATE('10:35','HH24:MI'));
INSERT INTO TRASEU_STATIE VALUES (101, 11, 1, TO_DATE('09:00','HH24:MI'),
TO_DATE('09:10','HH24:MI'));
INSERT INTO TRASEU_STATIE VALUES (101, 14, 2, TO_DATE('17:00','HH24:MI'),
TO_DATE('17:05','HH24:MI'));
-- Categorii tarif
INSERT INTO CATEGORIE_TARIF VALUES (1, 'Adult', 1.0, 18, 64);
INSERT INTO CATEGORIE_TARIF VALUES (2, 'Copil', 0.5, 0, 12);
INSERT INTO CATEGORIE_TARIF VALUES (3, 'Student', 0.7, 13, 25);
-- Tarife
INSERT INTO TARIF VALUES (100, 1, 50);  -- traseu 100, adult, 50 moneda
INSERT INTO TARIF VALUES (100, 2, 25);
INSERT INTO TARIF VALUES (101, 1, 300);
INSERT INTO TARIF VALUES (101, 3, 210);
-- Rezervări
INSERT INTO REZERVARE VALUES (5000, DATE '2025-05-29', 'card', 'activa');
INSERT INTO REZERVARE VALUES (5001, DATE '2025-05-28', 'numerar', 'activa');
-- Bilete
INSERT INTO BILET VALUES (9000, 5000, 1000, 1, DATE '2025-06-01', 50, 'A1',
```

```
'valid');
INSERT INTO BILET VALUES (9001, 5000, 1000, 2, DATE '2025-06-01', 25, 'A2',
'valid');
INSERT INTO BILET VALUES (9002, 5001, 1001, 1, DATE '2025-06-15', 300, 'B1',
'valid');
INSERT INTO BILET VALUES (9003, 5001, 1001, 3, DATE '2025-06-15', 210, 'B2',
'valid');
```

(Alte inserții similare se pot genera pentru a completa datele cerute.)

12. Cereri SQL complexe

a) Subcereri corelate (3 tabele): găsim biletele care au prețul peste media biletelor din aceeași rezervare:

```
SELECT b.BiletID, b.Pret, b.CategorieID
FROM BILET b
WHERE b.Pret > (
   SELECT AVG(b2.Pret)
   FROM BILET b2
   WHERE b2.RezervareID = b.RezervareID
);
```

Aici subinterogarea din `WHERE` *se corelează cu tabelul exterior* `BILET b` *(încluse în 3 tabele prin legături rezervare:bilet).*

b) Subcereri nesincronizate în FROM: calculăm totalul vânzărilor pe rezervare, apoi afișăm rezervările care au vândut peste 60 de unități:

```
SELECT R.RezervareID, R.DataRezervare, T.TotalBilete
FROM REZERVARE R
JOIN (
   SELECT b.RezervareID, COUNT(*) AS TotalBilete
   FROM BILET b
   GROUP BY b.RezervareID
) T ON R.RezervareID = T.RezervareID
WHERE T.TotalBilete > 60;
```

Aici subinterogarea (T) din `FROM` *nu se corelează cu exteriorul (este un „derived table” independent), urmând să fie legată prin* `JOIN` *cu* `REZERVARE`*.*

c) Grupări, funcții agregate, filtrare cu subcereri nesincronizate (3 tabele): calculăm încasările totale per traseu și păstrăm doar traseele cu încasări peste media încasărilor tuturor traseelor:

```
SELECT t.TraseuID, t.Denumire, SUM(b.Pret) AS IncasariTotale
FROM TRASEU t
JOIN TREN tr ON tr.TraseuID = t.TraseuID
JOIN BILET b ON b.TrenID = tr.TrenID
GROUP BY t.TraseuID, t.Denumire
HAVING SUM(b.Pret) > (
    SELECT AVG(sub.SumPret) FROM (
        SELECT SUM(b2.Pret) AS SumPret
        FROM BILET b2
        JOIN TREN tr2 ON tr2.TrenID = b2.TrenID
        JOIN TRASEU t2 ON t2.TraseuID = tr2.TraseuID
        GROUP BY t2.TraseuID
    ) sub
);
```

Aici sunt trei tabele (TRASEU, TREN, BILET). Filtrarea în `HAVING` *include o subinterogare nesincronizată (în SELECT din* `HAVING`*), calculând media sumelor per traseu.*

d) Ordonări + NVL + DECODE: ordonăm biletele descrescător după preț (înlocuind eventualele valori NULL cu 0 cu NVL) și folosim DECODE pentru a eticheta categoriile:

```
SELECT b.BiletID,
       NVL(b.Pret,0) AS PretAfisat,
       DECODE(b.CategorieID, 1, 'Adult', 2, 'Copil', 3, 'Student', 'Alta') AS
TipCategorie
FROM BILET b
ORDER BY NVL(b.Pret,0) DESC;
```

- **NVL(b.Pret,0)** înlocuiește prețul `NULL` cu 0 pentru sortare.
- **DECODE** transformă codul categoriei în text („Adult”, „Copil”, „Student” etc.).

e) Funcții pe șiruri, pe date și CASE: exemplu combinat:

```
SELECT b.BiletID,
       LOWER(SUBSTR(o.Nume,1,3)) AS PrefixOperator,
       ADD_MONTHS(SYSDATE, 1)      AS DataUrmLuna,
       EXTRACT(YEAR FROM b.DataCalatorie) AS AnCalatorie,
       CASE
         WHEN b.Pret > 100 THEN 'Scump'
         ELSE 'Normal'
       END AS ClasificarePret
FROM BILET b
```

```
JOIN TREN t ON b.TrenID = t.TrenID
JOIN OPERATOR o ON t.OperatorID = o.OperatorID;
```

- **LOWER**, **SUBSTR** (2 funcții pe șir): ia primele 3 caractere din numele operatorului și le pune cu litere mici.
- **ADD_MONTHS**, **EXTRACT** (2 funcții date): `ADD_MONTHS(SYSDATE,1)` calculează data peste o lună de azi; `EXTRACT(YEAR FROM b.DataCalatorie)` extrage anul călătoriei.
- **CASE**: clasifică biletul „Scump” sau „Normal” după preț.

f) Bloc cu WITH (Common Table Expression): calculăm numărul de bilete per traseu într-un subquery CTE și apoi afișăm traseele cu mai mult de 3 bilete vândute:

```
WITH BiletePeTraseu AS (
  SELECT tr.TraseuID, COUNT(*) AS NrBilete
  FROM TREN tr
  JOIN BILET b ON b.TrenID = tr.TrenID
  GROUP BY tr.TraseuID
)
SELECT t.TraseuID, t.Denumire, NVL(bp.NrBilete,0) AS NumarBilete
FROM TRASEU t
LEFT JOIN BiletePeTraseu bp ON t.TraseuID = bp.TraseuID
WHERE NVL(bp.NrBilete,0) > 3;
```

Aici `WITH BiletePeTraseu` *definește un tabel temporar cu statistici, iar interogarea principală îl folosește cu un LEFT JOIN pentru a filtra traseele.*

13. Operații UPDATE și DELETE cu subcereri

1. **UPDATE cu subinterogare:** actualizăm prețul biletelor scumpe (de ex. peste media tuturor biletelor) cu o reducere:

   ```
   UPDATE BILET
   SET Pret = Pret * 0.9
   WHERE Pret > (SELECT AVG(Pret) FROM BILET);
   ```

   Aici subinterogarea (`SELECT AVG(Pret)`) calculează media tuturor biletelor și actualizează doar biletele cu `Pret` mai mare.

2. **UPDATE cu subinterogare în SET:** de exemplu adăugăm 10 la toate trenurile ale unui operator anume:

   ```
   UPDATE TREN
   SET CapacitateLocuri = CapacitateLocuri + 10
   ```

```
WHERE OperatorID = (SELECT OperatorID FROM OPERATOR WHERE Nume =
'Metrorail');
```

Aici subinterogarea identifică operatorul după nume.

3. **DELETE cu subinterogare:** ștergem biletele foarte vechi (să zicem anterioare lui 2024):

```
DELETE FROM BILET
WHERE DataCalatorie < (SELECT MIN(DataCalatorie) FROM BILET) - 365;
```

(sau simplu folosind `DATE '2024-01-01'` direct). Subinterogarea oferă un prag dinamic.

În toate aceste exemple, subinterogările duc la filtrarea setului de rânduri înainte de efectuarea UPDATE/DELETE.

14. Vizualizare (VIEW) complexă și operații DML (LMD)

- **Creare view simplu (updatabil):**

```
CREATE OR REPLACE VIEW VTrenCap AS
  SELECT TrenID, CapacitateLocuri
  FROM TREN;
```

Această vedere simplă pe un singur tabel (`TREN`) permite actualizări.
Exemplu LMD permis:

```
UPDATE VTrenCap
SET CapacitateLocuri = CapacitateLocuri + 10
WHERE TrenID = 1000;
```

(Avertisment: Oracle permite în general `UPDATE` pe view-ul care corespunde exact unui tabel și include PK.)

- **Creare view complex (agregat/join – neupdatabil):**

```
CREATE OR REPLACE VIEW V_IncasariTraseu AS
  SELECT t.TraseuID, t.Denumire, COUNT(b.BiletID) AS NrBilete, SUM(b.Pret)
AS Incasari
  FROM TRASEU t
  JOIN TREN tr ON tr.TraseuID = t.TraseuID
  JOIN BILET b ON b.TrenID = tr.TrenID
  GROUP BY t.TraseuID, t.Denumire;
```

Această vedere realizează o agregare/join între 3 tabele.
Operație LMD permisă: de regulă **SELECT** pe aceasta (extragerea datelor).
Operație LMD nepermisă: **UPDATE/DELETE/INSERT** direct pe view – deoarece view-ul conține agregări/join, în Oracle nu permite modificări prin view. De exemplu:

```
UPDATE V_IncasariTraseu
SET NrBilete = NrBilete + 1
WHERE TraseuID = 100;
```

va fi respins (nu este un view updatabil).

Astfel, ca exemplu: **UPDATE** este permis pe `VTrenCap` (view simplu), dar **INSERT/UPDATE** nu este permis pe `V_IncasariTraseu` (view complex), demonstrând diferența dintre o vedere „updatabilă" și una nu.

15. Alte cereri SQL specializate

- **Outer join pe min. 4 tabele:** extragem informații complete pentru fiecare operator, chiar dacă nu are rezervări sau trenuri:

```
SELECT o.Nume AS Operator,
       NVL(t.NumarTren, '---') AS TrenNumar,
       NVL(b.Pret, 0) AS PretBilet,
       NVL(r.DataRezervare, DATE '1900-01-01') AS DataRezervare
FROM OPERATOR o
LEFT JOIN TREN t ON o.OperatorID = t.OperatorID
LEFT JOIN BILET b ON t.TrenID = b.TrenID
LEFT JOIN REZERVARE r ON b.RezervareID = r.RezervareID;
```

Acest query cu **LEFT JOIN** pe 4 tabele (Operator, Tren, Bilet, Rezervare) asigură că toți operatorii sunt afișați chiar dacă nu au bilete vândute. Funcția NVL tratează valorile lipsă (de ex. afișăm `0` sau `---` în loc de NULL).

- **Operația de diviziune:** găsim stațiile care apar în *toate* traseele existente (exemplu generic cu relația **Traseu_Statie**). Se caută `StatieID` care se asociază cu fiecare `TraseuID`:

```
SELECT s.StatieID
FROM STATIE s
WHERE NOT EXISTS (
  SELECT 1 FROM TRASEU t
  WHERE NOT EXISTS (
    SELECT 1 FROM TRASEU_STATIE ts
    WHERE ts.TraseuID = t.TraseuID
      AND ts.StatieID = s.StatieID
```

```
  )
);
```

Această formulă cu **NOT EXISTS**[op nesting] este echivalentul relațional al diviziunii: returnează stațiile `s` pentru care nu există niciun traseu `t` în care să nu apară `s`.

- **Top-N (analiză):** afișăm top 3 trasee după numărul de bilete vândute:

```
SELECT * FROM (
  SELECT t.TraseuID, t.Denumire, COUNT(b.BiletID) AS TotalBilete,
         RANK() OVER (ORDER BY COUNT(b.BiletID) DESC) AS Rnk
  FROM TRASEU t
  JOIN TREN tr ON tr.TraseuID = t.TraseuID
  JOIN BILET b ON b.TrenID = tr.TrenID
  GROUP BY t.TraseuID, t.Denumire
)
WHERE Rnk <= 3;
```

Aici folosim funcția analitică `RANK()` pentru a numerota traseele după numărul de bilete, apoi preluăm primele 3 (top-3).

16. Optimizarea unei cereri

Exemplu: presupunem următoarea interogare inițială, care folosește o subinterogare:

```
-- SQL inițial
SELECT r.RezervareID, r.DataRezervare
FROM REZERVARE r
WHERE r.RezervareID IN (
    SELECT b.RezervareID FROM BILET b WHERE b.Pret > 100
);
```

Expresie algebrică:
π_{RezervareID,DataRezervare} (σ_{RezervareID ∈ (select RezervareID din Bilet| Pret>100)} (Rezervare))

Putem optimiza prin transformarea subinterogării într-un JOIN:

```
-- SQL optimizat
SELECT DISTINCT r.RezervareID, r.DataRezervare
FROM REZERVARE r
JOIN BILET b ON b.RezervareID = r.RezervareID
WHERE b.Pret > 100;
```

Arbore algebric: (1) Se aplică selecția `σ_{Pret>100}` pe tabela BILET; (2) se realizează un *join* între rezultatul (1) și tabela REZERVARE pe `RezervareID`; (3) se proiectează coloanele dorite. Acest plan realizează filtrarea `Pret>100` *înainte* de join, reducând tuplurile.
În SQL optimizat, am evitat operatorul `IN` cu subselect și am folosit `JOIN`, ceea ce poate fi executat mai eficient de optimizer.

Astfel, **SQL anterior:** subinterogare în `WHERE`, **SQL optimizat:** `JOIN` + `DISTINCT`.

17. BCNF, 4NF, 5NF și denormalizare

- **BCNF (Boyce–Codd Normal Form):** extinde 3NF, cerând ca orice dependență funcțională X→Y să aibă X super-cheie. În cele mai multe proiecte practică, tabelele în 3NF satisfac și BCNF.
- **4NF:** elimină dependențele multivaluate. De exemplu, dacă ar exista un tabel în care două seturi de valori multiple sunt independente, ar trebui despărțit. (În baza noastră, de ex., dacă `TREN` ar avea listă de calificări pentru mecanici fără legătură cu vreun atribut de cheie, ar fi nevoie de 4NF.) Ronald Fagin a introdus 4NF în 1977 și 5NF în 1979 [6].
- **5NF (PJNF):** elimină dependențele de tip *join* (o relație care poate fi reconstruită prin join-ul mai multor relații) – se aplică rar, e mai teoretică.

Exemplu de justificare BCNF/4NF: Dacă am fi avut în `CATEGORIE_TARIF` un set mulțime de sub-categorii pentru fiecare categorie, conform BCNF ar trebui normalizat suplimentar. Dar de regulă, nu există astfel de dependențe complexe în modelul biletelor.

Denormalizare: uneori, pentru performanță, se acceptă redundanță controlată. De exemplu, am putea denormaliza prin includerea `Descriere` a categoriei în tabela `BILET`, pentru a evita join-ul în interogările de raport, dacă apare frecvent. Aceasta reduce costul de citire (mai puține join-uri) cu prețul introducerii de redundanță. Decizia de denormalizare se ia când interogările cititoare (raportare) sunt critice și viteză peste avantajele normalizării.

Bibliografie: Popescu I., Velcescu L. „Proiectarea bazelor de date” (2008); *Neprocedural în Oracle 10g* (2008). Informațiile teoretice despre modele conceptuale și normalizare sunt însoțite de definiții generale din surse academice [4] [2] și exemple din practică.

---

[1] [3] Online Railway Ticket Reservation System | GeeksforGeeks
https://www.geeksforgeeks.org/online-railway-ticket-reservation-system/

[2] Baze de date
https://info-hobby.ro/wp-content/uploads/2021/08/Baze-de-date-modele.pdf

[4] [5] [6] Database normalization - Wikipedia
https://en.wikipedia.org/wiki/Database_normalization